VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2012 Martin Střílka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Střílka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

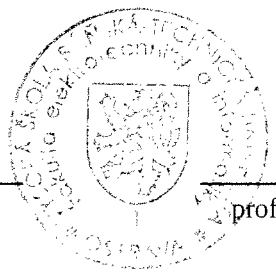
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Konzultant bakalářské práce: Bc. Ondřej Kvasnovský

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 30. dubna 2012 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2012 .....

Rád bych na tomto místě poděkoval všem kolegům z firmy Tieto Czech s.r.o., za jejich ochotu a odbornou pomoc. Zvláště pak Bc. Ondřeji Kvasnovskému za nabídnutou příležitost vykonání odborné praxe. Zároveň chci poděkovat mému vedoucímu doc. RNDr. Petru Šalounovi, Ph.D. za hodnotné rady.

Abstrakt

Bakalářská práce popisuje absolvování odborné praxe ve firmě Tieto Czech s.r.o. Bakalářskou odbornou praxi jsem vykonával od 6. října 2011 do 27. dubna 2012. Po celou dobu praxe jsem pracoval na informačním systému nazvaném MyWires, který měl sloužit pro správu ethernetových zásuvek v podnikové síti. Byl jsem zaměstnán na pozici softwarového vývojáře v týmu spolu se dvěma studenty, kteří také vykonávali odbornou praxi. Informační systém byl vyvíjen jako webová aplikace na platformě Java. Vytvořený systém je určen pro interní použití společností Tieto Czech. V současné době je systém téměř před dokončením a jeho nasazení brání pouze několik chybějících vlastností.

Klíčová slova: Java, Tieto, informační systém, odborná bakalářská praxe, webová aplikace

Abstract

Bachelor thesis describes professional practise in company Tieto Czech s.r.o. I was performing my bachelor professional practise from October 6, 2011 to April 27, 2012. I was implementing an information system called MyWires throughout the period of my practise. MyWires is intended to manage ethernet sockets in company network. I worked as software developer in team with other two students, who were also performing their professional practise. We were implementing this system as web application on the platform Java. This system is intended only for internal use by Tieto Czech. Now the system is almost finished, just lack of some minor functionality prevents from its deployment.

Keywords: Java, Tieto, information system, professional practise, web application

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
AOP	–	Aspect Oriented Programing
API	–	Application Interface
CSS	–	Cascade Style Sheets
CSV	–	Code Set Versioning
DAO	–	Data Access Object
DB	–	Databáze
DI	–	Dependency Injection
EJB	–	Enterprise Java Beans
GUI	–	Graphical user interface
GWT	–	Google Web Toolkit
HBM	–	Hibernate Mapping File
HTTP	–	Hyper Text Transfer Protocol
HTML	–	Hyper Text Markup Language
HQL	–	Hibernate Query Language
IT	–	Informační technologie
IoC	–	Inversion of control
JAXB	–	Java Architecture for XML Binding
J2EE	–	Java 2 Platform, Enterprise Edition
JavaSE	–	Java Standard Edition
LDAP	–	Lightweight Directory Protocol
ORM	–	Object Relation Mapping
OXM	–	Object/XML Mappring
POM	–	Project Object Model
POJO	–	Plain Old Java Object
RIA	–	Rich Internet Application
RPC	–	Remote Procedure Call
SOAP	–	Simple Object Access Protocol
SVN	–	Subversion
SWI	–	Softwarové Inženýrství
SQL	–	Structured Querry Language
TDD	–	Test Driven Development
TZD	–	Teorie Zpracování Dat

URI	–	Uniform Resource Identifier
VO	–	View Object
WS	–	Web Service
WSDL	–	Web Service Definition Language
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Svou odbornou praxi jsem vykonával ve společnosti Tieto Czech s. r. o. Pro tuto formu bakalářské práce jsem se rozhodl z toho důvodu, abych se lépe seznámil s reálným vývojem softwaru, abych si vyzkoušel práci v týmu a také abych zdokonalil své schopnosti vyvíjet software. Dostal jsem za úkol vytvořit informační systém v technologii Java, na kterém jsem se měl naučit danou technologii. Během praxe jsem si ověřil znalosti a dovednosti nabyté v průběhu studia. Zároveň jsem získal spoustu nových znalostí a vyzkoušel si jejich použití na reálných úlohách. Průběh odborné praxe s popisem řešených úkolů je uveden v této práci. Nejprve je stručně představena společnost, pracovní zařazení a použité technologie. Dále se zabývá popisem zadaných úkolů během praxe. K úkolu jsou uvedeny jaké postupy a technologie byly použity pro jeho řešení. Ke konci je popsán souhrn dovedností, které jsem nabyl během absolvování odborné praxe. V závěru bakalářské práce jsou shrnuty dosažené výsledky a její celkové zhodnocení.

2 Popis firmy Tieto a pracovního zařazení

2.1 Tieto

Společnost Tieto vznikla ve Finsku v roce 1999. Své současné jméno dostala v roce 2009. V začátcích zajišťovala vývoj a údržbu IT systémů především pro Finský Union Bank a její zákazníky a také pro lesní průmysl. Nyní je poskytovatelem IT služeb hlavně v severní Evropě, Německu a Rusku pro velké nebo středně velké organizace. Hlavními obchodními partnery jsou organizace z oblasti bankovnictví, telekomunikací a IT služeb. Tieto má pobočky ve zhruba 30 zemích světa, ve kterých zaměstnává cca 18000 odborníků. V průběhu minulého desetiletí se naplno projevila globalizace IT průmyslu a společnost rozšířila své mezinárodní působení. V roce 2004 otevřela první pobočku v České republice. Řadí se mezi nejvýznamnější společnosti podnikající v oboru IT služeb v Moravskoslezském kraji. Sídlo společnosti se nalézá v Ostravě, kde je zaměstnáno okolo 1800 lidí. [11]

2.2 Popis pracovního zařazení

Pro svou odbornou praxi jsem byl přiřazen do oddělení JOS Enterprise Solutions na pozici Software Developer. Byl jsem přidělen na interní projekt MyWires, který jsem měl implementovat spolu se dvěma kolegy studenty. V průběhu praxe se ovšem počet kolegů na projektu měnil, většinou nás bylo asi šest, nicméně od února jsme zůstali jen my tři. Zbylí 3 kolegové byli zaměstnanci Tieta. Po celou dobu vývoje nám byl po ruce kolega ing. Peter Almásy, který nám pomáhal jako analytik. Mým úkolem byl návrh a implementace částí systému. Můj přímý nadřízený byl pan Bc. Ondřej Kvasnovský, který nám projekt zadal, také nám zadával úkoly a určoval jejich priority. Zároveň spolu s Peterem vystupoval v roli zákazníka a hodnotil náš postup. Na vypracování úkolu jsme měli 50 dnů.

3 Úkoly zadané během odborné praxe

Prvním úkolem bylo seznámit se s technologiemi, které budeme používat v průběhu praxe. Pro náš projekt jsme měli použít technologii Java s frameworky Hibernate, Spring, Vaadin a s nástroji Eclipse, Maven a Subversion.

3.1 Projekt MyWires

Spolu s kolegy jsme dostali úkol na vytvoření informačního systému pro správu internetových zásuvek RJ-45. Tento systém má sloužit hlavně pro síťové specialisty. Nynější situace je taková, že v budovách firmy Tieto je množství internetových zásuvek, které jsou nezapojeny. Když zaměstnanec požádá o zapojení některé zásuvky, tak může nastat situace kdy síťový specialista je nucen odpojit jednu již zapojenou zásuvku. Tady nastává problém, jakou zásuvku může odpojit, když nemá žádnou evidenci používaných zásuvek. Při stěhování zaměstnanců k takovéto situaci může dojít poměrně často. Pro běžného zaměstnance tento systém přináší výhody v podobě přehledu o vlastních zásuvkách a také možnosti podání požadavku pro zapojení nové zásuvky.

Systém by se měl chovat jako Rich Internet Application (RIA), to znamená, že má některé vlastnosti desktopvé aplikace ale je spouštěn z webového prohlížeče bez nutnosti další instalace. Aplikace MyWires by měla být postavena na technologii JavaEE.

3.2 Funkční specifikace systému

Projekt jsme začali tvořit „na zelené louce“. Začali jsme prvními user stories, které popisují různé typy uživatelů a akce, které s tímto systémem mohou provádět. Podle toho se pak staví Use Case diagramy a doména. V této fázi jsem si vyzkoušel práci analytika.

3.2.1 Role

Uživatelé k systému mohou přistupovat ve čtyřech rolích. Uživatel může být ve více rolích současně, přičemž platí, že v jednom týmu má právě jednu roli.

3.2.1.1 Team Member Česky člen týmu, základní role, ve které se nejčastěji budou vyskytovat běžní zaměstnanci Tieta. Jsou k dispozici tři různé varianty, které se liší počtem práv. Uživatel s touto rolí může provádět tyto aktivity:

- prohlížet své zásuvky a požadavky pro zásuvky
- měnit nastavení profilu
- případně pouze na povolení vedoucího týmu
  - editovat nastavení zásuvek
  - vytvořit požadavek pro zapojení zásuvky
  - odstranit zásuvku
  - sledovat stav svých požadavků pro zapojení zásuvky

3.2.1.2 Team Leader Neboli vedoucí týmu, uživatel s touto rolí je zodpovědný za správu týmu. Pro každý tým existuje právě jeden Team Leader. Tato role dědí všechna práva role Team Member. Uživatel s touto rolí může provádět tyto aktivity:

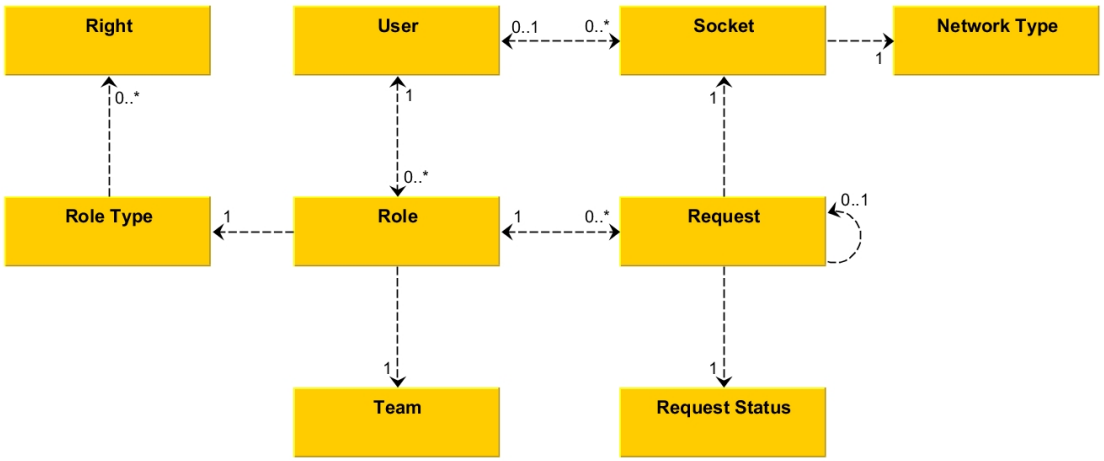
- přidávat členy týmu
- odebírat členy týmu
- měnit údaje o týmu
- nominovat uživatele, který ho může dočasně zastupovat v roli Team Leader
- potvrzovat požadavky členů týmu
- zamítat požadavky členů týmu
- přiřadit/odebrat uživateli práva pro vytvoření, odstranění a změnu svých zásuvek bez nutnosti potvrzení

3.2.1.3 Network Specialist Síťový specialista provádí fyzické zapojení zásuvek. Nedědí žádné právo z předchozích dvou rolí. Uživatel s touto rolí může provádět tyto aktivity:

- zobrazit všechny zásuvky v systému
- vyhledávat v zásuvkách podle následujících kritérií
  - typ sítě
  - budova
  - číslo zásuvky
  - tým
  - login uživatele

3.2.1.4 Administrator Administrátor je speciální role používaná členy týmu pro správu systému MyWires. Uživatel s touto rolí může provádět tyto aktivity:

- zobrazit seznam uživatelů aplikace
- zobrazit seznam typů sítě
- přidat uživatele do systému
- přidat tým do systému
- editovat profil uživatele
- přidat uživatele do týmu



Obrázek 1: Diagram doménových tříd

- změnit roli uživatele v rámci týmu
- přidat typ sítě

Uživatel s touto rolí nemůže provádět tyto aktivity:

- zobrazit nebo změnit data o připojení ostatních uživatelů

3.2.2 Use Case diagram

Diagram s případy užití je na obrázcích číslo 10 a 11 v příloze.

3.2.3 Doménový model

Diagram doménových tříd je na obrázku č. 1.

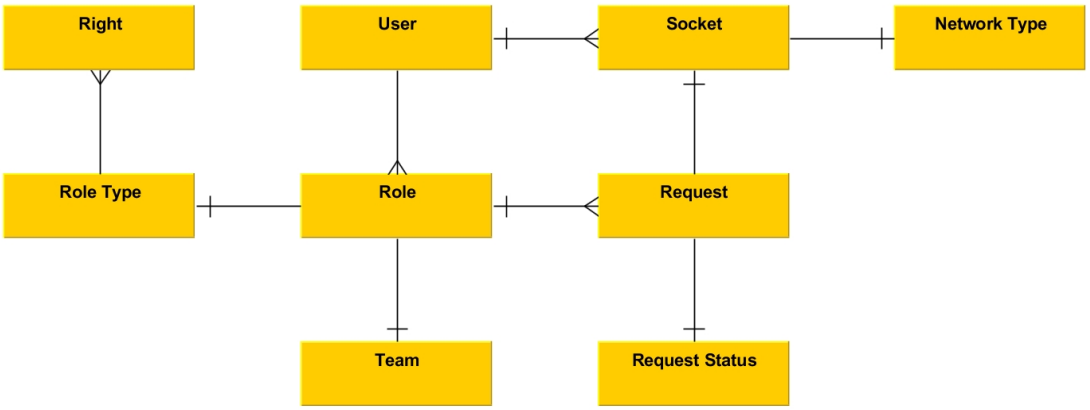
3.2.4 Entity Relationship diagram

Diagram je na obrázku č. 2, popis jednotlivých entit je v části textu č. 6.2.1.

3.3 Technická specifikace systému

3.3.1 Specifikace technologií

Systém bude rozdělen na Backend a Frontend, spojení těchto 2 částí bude realizováno webovými službami kdy Frontend bude klient a Backend bude server. Projekt bude dále rozdělen na moduly pomocí nástroje Maven. Pro objektově relační mapování bude použit framework Hibernate. Uživatelské rozhraní bude vytvořeno pomocí aplikačního



Obrázek 2: Entity Relationship Diagram

Jméno tabulky	Počet záznamů	Velikost záznamu [B]	Datová náročnost [kB]
Zásuvka	10000	50	500
Požadavek	6000	70	420
Role	4000	50	200
Uživatel	2000	200	400
Tým	400	40	16
Právo	50	30	1
Typ role	4	30	1
Typ připojení	3	30	1

Tabulka 1: Velikost ukládaných dat

rámce Vaadin. Vlastnosti aplikačního rámce Spring budou využívány v každém modulu. Databáze bude MySQL.

3.3.2 Náročnost na databázi

Tabulka č.1 ukazuje náročnost jednotlivých tabulek na úložiště. Údaje v ní zobrazené by měly odpovídat běžnému provozu a jsou pouze orientační. Celková náročnost databáze na uložená data je cca 1,5 MB.

3.3.3 Náročnost na síť

Jednotky až desítky uživatelů přihlášených současně, z toho plynou zhruba desítky dotazů za minutu. Běžná velikost odpovědi posílané uživateli bude cca 5kB. Při desíti připojených uživatelích datový tok může dosahovat 50kB/s, to by měla zvládat jak spojení mezi Backendem a Frontendem, tak i spojení z Frontendu do vnitřní sítě Tieta. Mezi částmi Frontend a Backend by délka odezvy v síti neměla překračovat 100ms.

4 Technologie

Protože nezbytnou podmínkou pro implementaci projektu byla znalost použitých technologií, tak je alespoň stručně představím.

4.1 Maven

Maven je nástroj pro automatizaci a správu sestavování („build“) aplikace. Nejčastěji se používá pro projekty v Javě, ale lze použít i pro projekty v C# nebo Ruby. Usnadňuje sestavování projektu a zavádí na to jednotný systém. Maven používá XML pro popis projektu se všemi jeho závislostmi na externích knihovnách, pluginech, modulech, pořadí modulů při sestavování projektu a s dalšími různými funkcemi, například spouštění testů. Tento soubor se jmenuje Project Object Model (`pom.xml`) a je umístěn do kořenového adresáře projektu. Každý modul má také svůj vlastní soubor `pom.xml`, který dědí od nadřazeného souboru `pom.xml` a přidává své atributy. Díky tomuto rozvržení je možné celý projekt sestavit jediným příkazem. Maven všechny knihovny automaticky stahuje z veřejného globálního repozitáře, ovšem je možné si pro potřeby projektu založit i soukromý repozitář. Více se dočtete v [3].

4.2 Spring

Spring je aplikační rámec („framework“) pro vývoj J2EE aplikací. Spring může být použit pro libovolnou aplikaci jako alternativa k EJB. Hlavní návrhové vzory, které Spring používá, jsou Inversion of Control a Dependecy Injection. Jádro Springu se nazývá IoC kontejner. Inversion of Control přesouvá zodpovědnost za vytváření provázání objektů z aplikace na framework. Dependency Injection řeší vkládání objektů. Objekty, které IoC kontejner vytvoří, se nazývají Beany a jsou v něm uchovávány. Aby Spring věděl, které objekty vytvářet a kam je vkládat, musíme Beany definovat v konfiguračním souboru.

Spring se skládá asi z 20 modulů, které jsou rozděleny do 6 hlavních skupin. Jsou to Core, který obsahuje kontejner, dále Data Access skupina poskytující vrstvu pro ORM a Objektové/XML mapování. Web poskytuje základní webově orientované funkce. AOP a Instrumentation moduly obsahují implementaci pro podporu aspektově orientovaného programování. Skupina Test podporuje testování komponent Springu pomocí JUnit.

Spring odstraňuje těsné vazby jednotlivých POJO objektů a vrstev pomocí návrhového vzoru Inversion of Control. Odděluje konfiguraci od kódu, což zaručuje, že konfigurace je na jednom místě a není roztroušena ve zdrojových kódech. Tyhle prvky považuji za hlavní výhody Springu. Více o Springu se dočtete zde [1].

4.3 Hibernate

Hibernate je framework používaný pro perzistentní vrstvu a poskytující Object Relation Mapping (ORM). Jinými slovy mapování objektově orientovaného doménového modelu na entity relační databáze. Datové typy Javy mapuje na datové typy relační databáze. Hibernate řeší nekompatibilitu Java objektů a entit tím, že pro přístup k relační databázi

používá objekty vyšší úrovně. Tyto objekty jsou schopny generovat výsledné SQL dotazy, což také umožňuje snadnou výměnu DB.

Mapování Java objektů na DB entity se konfiguruje pomocí XML souborů nebo pomocí anotací. Z těchto informací se generuje DB schéma. Hibernate poskytuje perzistenci POJO objektů, jedinou podmínkou je bezparametrický konstruktor. Kromě POJO objektů lze mapovat i dědičnost, jsou možné tři způsoby. Pro každou třídu jedna tabulka, tabulka pro celou hierarchii nebo tabulka pro každou třídu ale pouze se sloupci, o které rozšiřuje svého předka. Pro zrychlení načítání implementuje návrhový vzor Lazy Load. Pro příbuzné objekty je možno nakonfigurovat kaskádu (Cascade), kterou zajistíme konzistenci dat při mazání a vytváření záznamů. Příklad pro použití kaskády by mohl být model album s kolekcí písní, když mažu album tak chci smazat i písně, které album obsahuje. Pro zrychlení aktualizace perzistovaných objektů si Hibernate pamatuje změněné hodnoty členských proměnných a pouze ty aktualizuje do DB.

Pokud potřebujeme sestavit SQL dotaz, máme k dispozici tři způsoby. V preferovaném pořadí to jsou Criteria, HQL dotaz a SQL dotaz. Criteria jsou objektové a vždy validní proto preferované, dotaz sestavený pomocí HQL je také vždy validní. Ručně napsaný SQL dotaz už nemusí být vždy validní v době kompilace, proto se používá jako krajní řešení.

Výhodou Hibernatu je automatické ORM a jednoduchost s jakou se používá. Hlavní nevýhoda je jeho rychlost, oproti staticky napsanému ORM je mnohem pomalejší. Nicméně výhody převažují. Více o Hibernatu se dočtete v publikaci [4].

4.4 Vaadin

Vaadin je „open source“ aplikační rámec pro platformu Java. Je určen pro tvorbu webových aplikací typu Rich Internet Application (RIA). Používá se pro vytváření uživatelského rozhraní.

Vaadin je rozdělen na serverovou a klientskou část. Serverová část obstarává logiku aplikace, s klientskou částí komunikuje pomocí technologie AJAX. Klientská část je postavena na technologii Google Web Toolkit (GWT). Kód aplikace je psán v Javě, který je vykonáván na serveru a pomocí GWT překládán do JavaScriptu, který je interpretován v prohlížeči. To považuji za největší výhodu Vaadinu, protože to umožňuje, aby programový model byl blíže softwarovému vývoji GUI než u tradičních webových aplikací. Vývojář také nemusí věnovat své úsilí JavaScriptu a HTML. Spuštění aplikace na klientské straně nevyžaduje žádný dodatečný plugin ani Javu, stačí mít prohlížeč podporující HTML a JavaScript. Díky použití GWT je zaručena kompatibilita s nejpoužívanějšími prohlížeči, takže vývojář nemusí psát aplikaci zvlášť pro jednotlivé prohlížeče.

Tento aplikační rámec poskytuje základní komponenty pro tvorbu GUI, styl práce s nimi je podobný jako s komponentami knihovny Swing. Základní komponenty mohou být rozšířeny vlastními GWT „widgety“. Vzhled komponent je možno upravovat pomocí CSS. Vaadin aplikace se chová jako Java Servlet a může být nasazena na libovolný aplikační server podporující Javu. Více o aplikačním rámci Vaadin se dočtete v publikaci [2].

4.5 Jednotkové a integrační testy

Pro testování kódu jsme používali JUnit testy s použitím aplikačních rámců Mockito a PowerMock, dále pro testování integrity byly napsány integrační testy.

JUnit je aplikační rámec pro jednotkové testy v jazyce Java. JUnit je základním kamenem vývojových technik Test Driven Development a Extrémního Programování. JUnit testy neslouží pro testování integrity aplikace, pouze ověřují správné výsledky při různém použití metody. [5]

Mocktito je testovací aplikační rámec, který dovoluje vytvářet „mockované objekty" pro použití v JUnit testech. Jsou to pouze falešné objekty a můžeme jim přesně definovat chování při volání konkrétních metod. Což umožňuje testovat pouze tu jednu konkrétní třídu a ne ty, které používá ve svých metodách. Proto je vhodný pro použití v JUnit testech nikoli v integračních testech. Další informace o Mockitu naleznete zde [6].

Tam, kde Mockitu „dochází dech", byl použit PowerMock, který umožňuje „mokovat" privátní metody, finální třídy, konstruktory atd.

Pro testování provázání částí systému jsou použity integrační testy. Slouží pro kontrolu toho, že nově přidané funkcionality nekolidují se stávajícími a chovají se stejně jako během jednotkových testů. Pro tyto testy je využita funkcionalita Springu.

4.6 Webová služba

Webová služba umožňuje komunikaci mezi dvěma počítači v síti a je popsaná jazykem WSDL (Web Services Description Language). WSDL popisuje to, co služba nabízí za procedury a způsob volání jednotlivých procedur. Pro komunikaci se používá protokol SOAP (Simple Object Access Protocol). SOAP je protokol pro komunikaci založenou na XML zprávách a jako aplikační vrstvu používá HTTP protokol. SOAP zprávy se přenášejí pomocí HTTP protože firewally již zavedených systémů nemají dostatek důvěry v SOAP. Protokol umožňuje různá schémata komunikace, například jednosměrnou komunikaci nebo požadavek/odpověď.

Náš systém používá schéma jménem RPC (Remote Procedure Call), kde jeden z účastníků je klient a jeden a druhý je server. Klient vytváří požadavky („request") a server mu posílá odpovědi („response"). Požadavky i odpovědi mají pevně definované parametry.

Nevýhodou XML základu služeb je velký zápis komunikace a parsrování velkého množství textu. Oproti tomu je binární zápis kratší ale nečitelný pro člověka. Více informací naleznete zde [10].

5 Agilní metodiky vývoje softwaru

Protože práci v týmu jsem si vyzkoušel poprvé a také jsem nikdy nepracoval na tak rozsáhlém projektu, rozhodl jsem se přiblížit vývoj softwaru v týmu a agilní metodiky vývoje. Specifikace našeho projektu se dost často měnila, proto byly zvoleny agilní metodiky vývoje. Klasický vodopádový model vývoje by byl nevhodný.

5.1 SCRUM

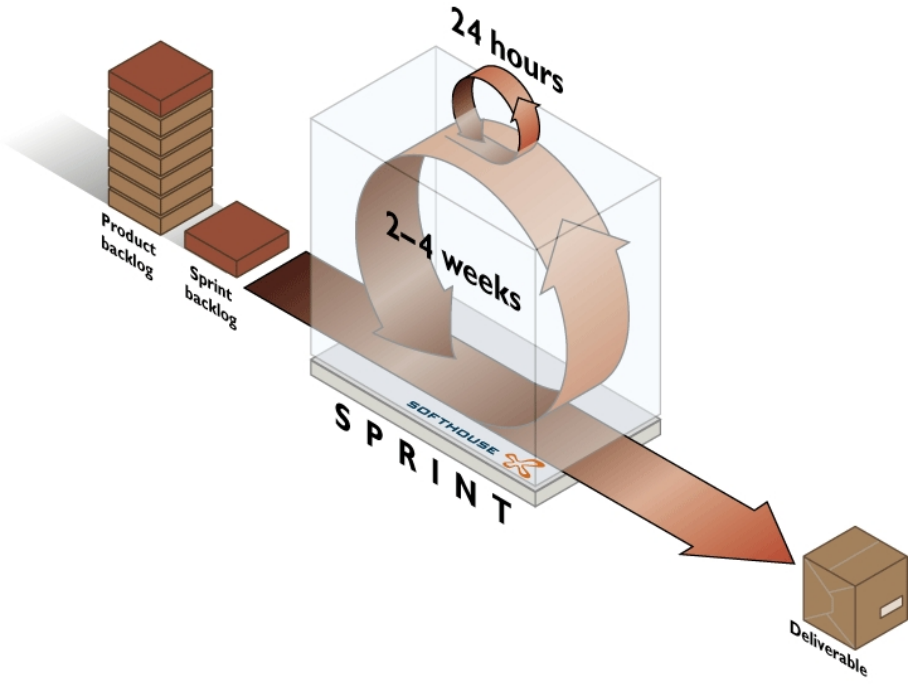
V praxi se zadání projektu často mění, proto se používají Agilní metodiky vývoje, které se dokážou změnám lépe přizpůsobit. Na začátku práce na projektu je představena vize finálního produktu. Sepisují se User Stories, které Product Owner seřadí podle priority do Product Backlogu. Product Owner je osoba odpovědná za to co se bude v příštím sprintu implementovat a určuje prioritní úkoly. Product Backlog je seznam všech požadovaných vlastností projektu. Každému úkolu se přiřadí časová náročnost. Další významnou rolí je Scrum Master, který odstiňuje vývojáře od okolního světa, řídí tým a řeší konflikty. Scrum Master může být Analytik nebo Project Manager, ale nemůže být vývojářem, protože by sám nebyl odstíněn. Metodika SCRUM je vhodná pro týmy o velikosti čítající čtyři až patnáct vývojářů. [7, 8]

Pro podporu Agilních Metodik existují různé nástroje, náš tým používal software Jira. Který umožňuje snadnou správu úkolů, přechodů mezi iteracemi atd.

5.1.0.1 Iterace U této metodiky je nazývána sprint. Délka iterace je pevně daná, zpravidla bývá dva týdny až měsíc. Iterace se skládá z analýzy, implementace a předání zákazníkovi. Na začátku sprintu se analyzují požadavky pro danou iteraci a z analýzy se sestaví jednotlivé úkoly spolu s časovou náročností. Ty se zapíšou do Sprint Backlogu. Vývoj probíhá dva až čtyři týdny, poslední týden se už nepřidává nová funkcionalita, pouze se odlaďuje stávající kód. V tomto posledním týdnu projekt přejde do stavu „code-freeze“. To proto, aby při prezentaci zákazníkovi software nepadal a choval se tak jak má.

5.1.0.2 Stand-up meeting Každodenní setkání ve stoje neboli Stand-up meeting. Konají se v dopoledních hodinách, naše setkání začínalo vždy v 9:30. Účastní se ho členové týmu a Scrum Master. Každý člen týmu řekne tři věci, co dělal včera, co má v plánu dělat dnes a jaké nastaly problémy při práci, které mu brání v pokračování.

5.1.0.3 Prezentace zákazníkovi neboli demo Pro demo se připraví scénáře, na kterých se předvede nově implementovaná funkcionalita. Vlastnosti, které nejsou dokončeny, se neukazují. Hodnotí se předchozí iterace, pokud například zbyly nějaké úkoly v Sprint Backlogu, tak se přesunou do další iterace. Naplánují se a stanoví cíle pro příští iteraci, z Project Backlogu se vytáhnou další úkoly podle priority. Zákazník hodnotí produkt a má možnost změnit zadání podle svých potřeb.



Obrázek 3: SCRUM [9]

5.1.0.4 Práce v týmu a sdílení kódu Pro sdílení kódu jsme používali nástroj Subversion. Jedná se o systém pro správu a verzování souborů, v našem případě zdrojových kódů. Řadí se mezi Version control nástroje a je obdobou systému CSV. Skládá se ze dvou hlavních částí, serverové klientské. Na serverové části je centrální úložiště verzovaných souborů („repository“), klientská část slouží pro práci s verzemi v souborovém systému. Vývojář provádí změny kódu nejprve u sebe lokálně a pak je nahraje na server („commit“). Každá změna vytvoří novou revizi celého repozitáře. Revize obsahuje informace o tom, co bylo změněno, kdo změnu provedl, čas a komentář. Problém nastává když nahrávaný soubor je v repozitáři v jiné revizi než v lokálním repozitáři, to znamená, že byl změněn nějakým jiným uživatelem. V tom případě se změny musí sloučit („merge“).

Pro práci s klientskou částí SVN jsme používali program TortoiseSVN.

6 Zvolený postup řešení zadaných úkolů

Systém jsme implementovali v technologii Java. Projekt měl hlavně sloužit pro naučení mě a mých kolegů, z tohoto důvodu byly použity webové služby, které systém rozdělují na Frontend a Backend. Projekt je rozdělen do několika modulů pomocí nástroje Maven. Ve fázi implementace jsem pracoval v roli vývojáře a testera zároveň.

6.1 Architektura systému

Projekt je rozdělen na Frontend část a Backend část. Spojení je realizováno přes webové služby, které využívají protokol SOAP. Naše služby používají šablonu RPC (Remote Procedure Call), kde jeden z účastníků je klient (Frontend) a jeden je server (Backend). Toto rozdělení umožňuje, aby třeba uživatelské rozhraní bylo napsáno v jiné technologin než Java nebo by mohly vedle sebe existovat dvě různá uživatelská rozhraní aplikace. Například tenký a tlustý klient. Jednotlivé části jsou dále rozděleny na moduly (obrázek č. 4). V této kapitole se budu věnovat popisu modulů systému.

6.2 Backend

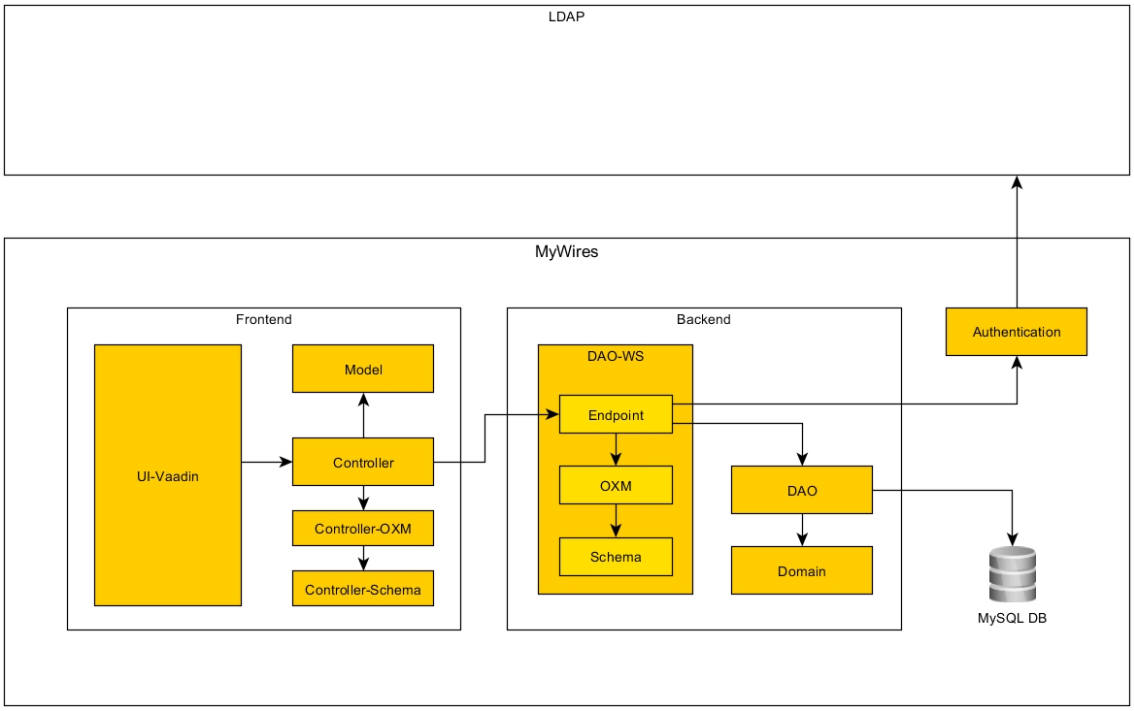
Tato kapitola je věnována popisu vlastností jednotlivých modelů v Backend části systému.

6.2.1 Domain

V modulu Domain jsou definovány doménové třídy aplikace. Doménový model tvoří srdce každé aplikace, a proto je na něj kladen velký důraz při návrhu. Náš model není příliš rozsáhlý, obsahuje pouze osm tříd (obr. č. 1). Ovšem obsahuje cyklus mezi entitními typy $User$, $Socket$, $Request$ a $Role$, který komplikuje použitelnost a dal by se označit za chybu v návrhu. Pokud je cyklus nepřerušen tzn. žádná z jeho vazeb nemá hodnotu null, tak při mapování na OXM objekty dojde k zacyklení a mapující objekt vyhodí výjimku. Pro tyto případy jsou implementovány metody jako $removeCycleInObjectGraph$, které předcházejí zacyklení. Toto řešení je ale nepěkné, daleko lepší by bylo odstranit vazbu mezi entitami $User$ a $Socket$ v doménovém modelu. Bohužel na tuto úpravu už nezbyl čas.

Třída $Right$ je výčtový typ obsahující jednotlivá práva. Například práva pro zobrazení, editování, odstraňování vlastních požadavků. $RoleType$ je zde pro jednotlivé typy rolí. Třída $Socket$ zapouzdřuje informace o ethernetové zásuvce, například číslo zásuvky a číslo místnosti. Typ sítě zásuvky je ve výčtovém typu $NetworkType$. Třída $User$ v sobě nese informace o uživateli, jako jsou jméno a příjmení nebo dodatečné informace o zobrazování historie, preferovaném barevném téma systému atd. Heslo k účtu neobsahuje, protože autentizace uživatele se provádí přes LDAP. Informace o týmu jako název a popis jsou zapouzdřeny v třídě $Team$.

Hlavní nositelé informací jsou třídy $Role$ a $Request$. Uživatel k systému vystupuje pod různými rolemi v různých týmech a platí, že může mít jen jednu roli v jednom týmu. Máme čtyři různé typy rolí ($RoleType$), které se liší svými právy (viz. specifikace

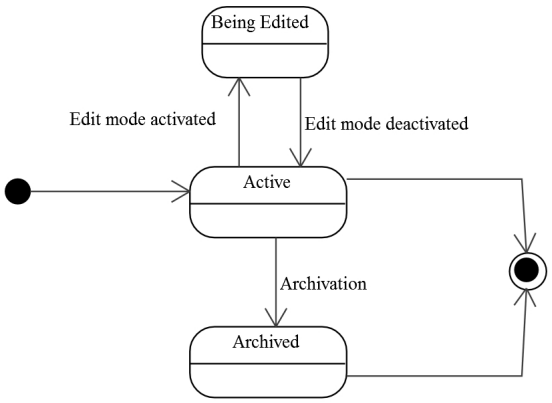


Obrázek 4: Modulární architektura systému MyWires se závislostmi modulů

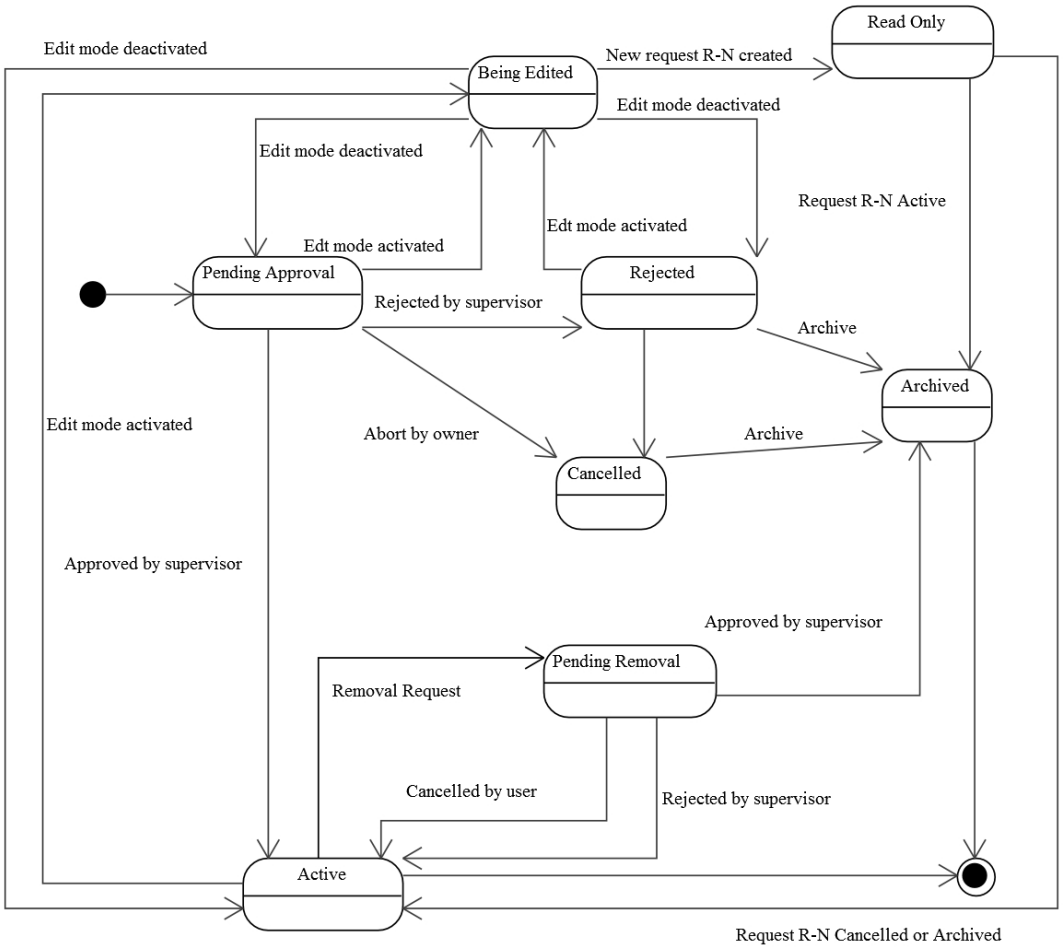
projektu na stránce č. 6). Uživatel jako takový žádné práva nemá. Třída $Request$ slouží pro požadavky uživatele, podle stavu požadavku ($RequestStatus$) se rozlišuje, jestli je zásuvka zapojena, čeká na zapojení, jestli byl požadavek zamítnut atd. Životní cykly požadavku se liší tím, že pokud uživatel v roli, přes kterou požadavek vytváří, má právo $Auto - Approve$. Toto právo má každý $TeamLeader$, $TeamMember$ ho může, ale nemusí mít. Pokud $TeamMember$ má $Auto - Approve$ právo, tak jeho požadavky nemusí schvalovat $TeamLeader$ ale jsou automaticky schváleny. Více v obrázku číslo 5.

Pokud $TeamMember$ právo $Auto - Approve$ nemá (obrázek č. 6), tak vytvoření nebo odstranění požadavku musí být schváleno $TeamLeaderem$. Vytvořený požadavek čekající na schválení je ve stavu $PendingApproval$, po schválení přejde do stavu $Active$, který označuje zapojené a používané zásuvky. Při zamítnutí požadavek přejde do stavu $Rejected$. Uživatel má možnost neschválený požadavek odstranit, tím přejde do stavu $Cacelled$. Ze stavů $Rejected$ a $Cancelled$ po jednom týdnu požadavky přecházejí do stavu $Archived$. Tento přechod je zajištěn databázovou procedurou. Pokud uživatel chce odstranit aktivní požadavek, tak nejprve přejde do stavu $PendingRemoval$ a čeká na schválení od vedoucího. Pokud je schválen, tak požadavek není přímo odstraněn, ale přejde do stavu $Archived$ a uživatel jej při implicitním nastavení neuvidí. Při odmítnutí přejde zpět do aktivního stavu.

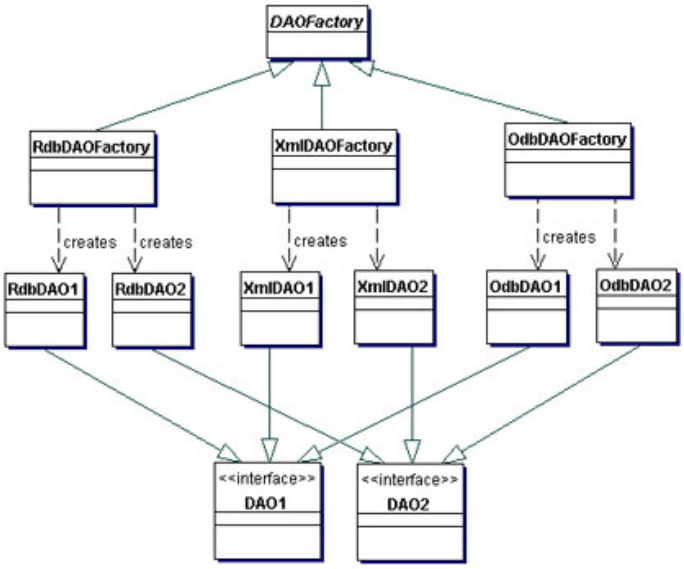
Speciální případ je změna požadavku, pokud je editován požadavek čekající na schválení, tak se změna potvrzovat nemusí. Ale pokud je požadavek v aktivním stavu, tak



Obrázek 5: Životní cyklus požadavku s právem Auto-Approve



Obrázek 6: Životní cyklus požadavku bez práva Auto-Approve



Obrázek 7: DAO Factory

změnu musí potvrdit vedoucí týmu. V takovém případě se vytvoří kopie původního požadavku se stavem $PendingApproval$. Původní požadavek přejde do stavu pouze pro čtení ($ReadOnly$), aby při zamítnutí nebyla ztracena původní informace. Po schválení se stav kopie změní na aktivní a původní požadavek se změní na archivovaný. V opačném případě archivovaný bude kopie a původní požadavek bude opět aktivní. Když uživatel začne editovat požadavek, tak aby se předcházelo různým konfliktům, změní se jeho stav na $BeingEdited$ v databázi. Takový požadavek není možné schválit ani zamítnout. Mohlo by se například přihodit, že uživatel začne editovat požadavek a než změny uloží, tak vedoucí týmu může tentýž požadavek schválit nebo odmítnout. Kdyby pak chtěl uživatel změny uložit, tak v ten moment by došlo ke kolizi a v lepším případě by vše skončilo vyhozením výjimky a chybovou hláškou. V horším případě by došlo k nekonzistenci dat.

Dokonce i v modulu, kde jsou jen definovány doménové třídy, máme jednotkové testy, které testují set a get metody.

6.2.2 DAO

Modul DAO používá aplikační rámec Hibernate spolu se Springem pro přístup k databázi. Třídy pro práci s Hibernatem jsou implementovány pomocí návrhového vzoru Data Access Object (DAO). Tento vzor pro každou doménovou třídu vytváří jednu DAO třídu, která se stará o všechny databázové operace prováděné pouze s daným typem doménového objektu (obrázek č. 7)[12]. Doménové třídy zároveň slouží jako Transfer Objekty. Vytváření DAO objektů je přenecháno třídě $DaoFactory$, která implementuje návrhový vzor Factory. Metody DAO tříd jsou deklarovány v rozhraních, to proto, aby při různých způsobech ukládání perzistovaných objektů bylo možné provádět stejné operace.

Knihovny Hibernate nepoužíváme přímo ale právě s využitím Springu. To zjednodušuje použití a také umožňuje používat prvky AOP (Aspect Orientd Programing), kdy metody v DAO třídách nemusíme rozšiřovat o například kód pro záznam průběhu vykonávání metod (log), ale jen je označíme anotací pro log a tento kód doplní až kompilátor. Samotný kód pro záznam do logu je zapouzdřen ve třídě $LoggingInterceptor$. Tento postup umožňuje zapouzdření takzvaných průřezových kódů.

Perzistované objekty jsou specifikovány v `*hbm.xml` souborech, ve kterých jsou také uložené HQL dotazy. Tyto definice odrážejí doménu a Hibernate podle nich generuje databázové schéma. Každá třída má nastavené aktualizování pouze změněných atributů. Pokud třída má jako atribut kolekci objektů, tak pro kolekci je nastaven Lazy Load. To znamená, že kolekce není načítána z databáze v době načítání objektu ale až když je použita. Ostatní atributy mají nastavení „eager“, to znamená, že jsou načítány v době načítání objektu.

Tento modul má téměř 100% pokrytí jednotkovými a integračními testy. Jednotkové testování DAO tříd ve většině případů není nutné, protože obsahují pouze jednoduché metody. Testy jsou zde v rámci TDD (Test Driven Development). Mnohem větší smysl zde mají integrační testy, které hlídají správné provázání s databází.

6.2.3 DAO-WS

Modul DAO-WS zapouzdřuje tři submoduly pro webové služby.

6.2.3.1 Moduly Schema a OXM Jsou v obou blocích stejné a definují kontrakt webové služby. Modul Schema definuje podobu jednotlivých služeb a tříd popisem v XML souborech. Jsou zde definice pro jednotlivé služby v podobě požadavku („request“) a odpovědi („respons“). Jsou zde i obdoby doménových tříd, protože doménové objekty nelze přímo převést do XML podoby. Pomocí JAXB jsou ze schémat vygenerovány OXM třídy do modulu OXM. Tyto třídy obsahují dodatečné informace pro mapování z OXM objektů na XML objekty a naopak. JAXB se stará i o zpětné převádění OXM objektů na XML objekty.

Na straně Backendu pro mapování doménových objektů na OXM objekty se používá třída třetí strany jménem Dozer Mapper, která je nakonfigurována tak, že mapuje atributy jednoho objektu na atributy druhého objektu podle názvu atributu. Přenos doménového objektu přes službu tedy vypadá následovně: doménový objekt → OXM objekt → XML objekt.

6.2.3.2 Modul Endpoint Definuje jednotlivé služby (procedury) a pracuje s DAO modulem pro manipulaci s daty. Obsahuje implementace jednotlivých služeb. Právě tento modul se nasazuje na aplikační server a poskytuje webové služby.

V balíčku $Security$ se nacházejí třídy pro autentizaci uživatele. Při každém vzdáleném volání procedury přes webové služby je nejprve ověřena identita uživatele, tak že v parametru každého dotazu je posílán objekt $Credentials$, který nese přihlašovací údaje uživatele. Autentizace se provádí přes modul Autentification. Pokud autentizace selže, není uživateli dovoleno volání procedury a je vyhozena výjimka.

Každý doménový objekt, který vstupuje nebo vystupuje z webové služby je ověřen pomocí validátorů v balíčku $Validators$. Ty například hlídají, jestliže objekt $Request$ má vyplněnou vazbu na objekt $Socket$ a podobně. Pokud je zaznamenána chyba v objektu, validátor vyhodí výjimku.

Implementace vzdálených procedur se nacházejí v balíčku $Service$. Procedury jsou rozděleny do tříd a každá je označená anotací $@Transactional$. To znamená, že její metody jsou transakční a provedou operaci „rollback“ pokud je vyhozena výjimka $ServiceException$ proto, aby nedošlo k nekonzistenci dat. V balíčku $Endpoint$ jsou definovány jednotlivé služby s využitím Spring WS API.

Tento modul má také téměř 100% pokrytí jednotkovými i integračními test. Tady už jsou třídy s některými složitějšími metodami, takže jednotkové testy pro ně byly nutné na rozdíl od modulu DAO. Pro integrační testy platí totéž jako v DAO modulu.

6.2.4 Authentication

Autentizace uživatele se provádí přes samostatný modul Authentication, který je napojený na systém LDAP. Tento modul je vně systému, protože je sdílen mezi více projekty. Pokud se uživatel může přihlásit do libovolného systému uvnitř firemní sítě, může se přihlásit i do systému MyWires.

6.3 Frontend

Tato kapitola je věnována popisu vlastností jednotlivých modelů v Frontend části systému.

6.3.1 Model

V modulu Model je Frontendová obdoba doménových objektů. Těmto objektům se říká View Objekty (VO). Tato kopie doménových objektů je zde pro lepší oddělení Frontedu a Backendu aplikace. Doménové objekty se používají pouze v Backendu a VO objekty pouze ve Frontendu.

6.3.2 Controller-Schema a Controller-OXM

Moduly Controller-Schema a Controller-OXM jsou stejné jako moduly OXM a Schema v Backend části. Tyto moduly musí být stejné, protože definují rozhraní WS.

6.3.3 Controller

Modul Controller odstiňuje Vaadin od WS. Obsahuje jednotlivé klienty („controllers“) WS. Tento modul se stará také o mapování OXM objektů na VO objekty. Tentokrát nepoužíváme žádné mapovací třídy třetích stran ale vlastní implementace mapovacích tříd. Objekt, který přijde přes webovou službu se mapuje následovně: XML objekt $\rightarrow$ OXM

objekt → VO objekt. V souboru `mywires-controller-ws-context.xml` jsou konfigurace klientů webových služeb. Každý klient má svoje vlastní URI. Klienti jsou rozdělení podle stránek aplikace, například $LoginPageController$ je určen pro přihlašovací stránku.

V tomto modulu už pokrytí kódu testy je okolo 70%. Mapovací třídy jsou otestovány jednotkovými testy kompletně, ale nejsou dopsány testy pro třídy klientů. Chybí hlavně integrační, které zde testují integritu webových služeb.

6.3.4 Vaadin-UI

Vaadin-UI je modul obsahující grafické rozhraní aplikace. Pouze v tomto modulu se pracuje s aplikačním rámcem Vaadin. Hlavní třídou je zde $MyWiresApplication$, která se chová jako servlet a lze ji nasadit na aplikační server vykonávající Javu.

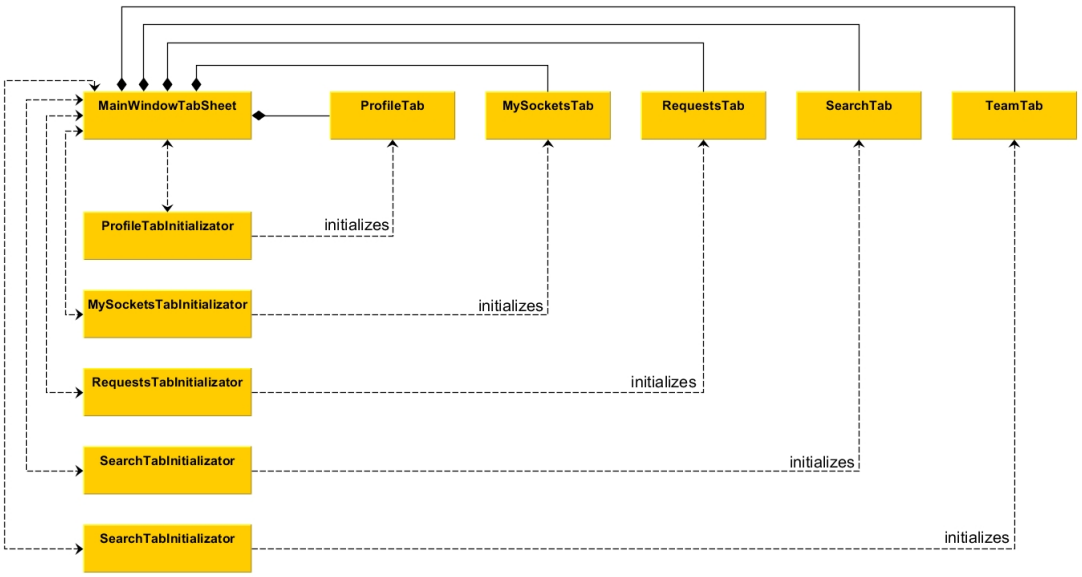
Je implementováno pět základních obrazovek. MySockets pro správu vlastních požadavků, Request pro schvalování/zamítání požadavků vedoucím týmu, Search pro vyhledávání aktivních zásuvek, Team pro vkládání/odebírání uživatelů z týmů. Profil pro zobrazování informací o přihlášeném uživateli a pro nastavení některých dodatečných funkcí systému. Tyto obrazovky rozšiřují Vaadin třídu $Panel$, který obohacují o své komponenty.

Tabulky v MySocket a Team obrazovkách jsou poměrně komplikované, proto se pokusím přiblížit jejich implementaci. Tabulky jsou výjimečné v tom, že poslední a editované řádky obsahují jiné komponenty než ostatní (viz. obrázek č. 12). Jsou to komponenty typu $ComboBox$ a $TextField$ a slouží pro datový vstup od uživatele. Poslední řádek umožňuje vytvářet nové požadavky. Editovaný se od něj moc neliší, a jak název napovídá, umožňuje měnit existující požadavky. Vaadin tabulka má pevně definované typy jednotlivých sloupců, třeba typy $String$ nebo $Button$. Problém nastává, až když chceme ve stejném sloupci mít na různých řádcích různé typy komponent. Vaadin nabízí řešení v podobě rozhraní $TableFieldFactory$. Při vkládání každé buňky na řádek používá třídu $TableFieldFactory$. V našem případě třídu $CustoTableFieldFactory$, která implementuje rozhraní $TableFieldFactory$. Zavolá její metodu $createField$, které předá id řádku a sloupce. Podle id řádku pozná, jestli se jedná o normální řádek, nový nebo editovaný. Tato metoda vrací objekt typu $Field$, což naše komponenty splňují, a podle id sloupce vrátí buď $ComboBox$ nebo $TextField$. Metoda $createFiel$d se volá na všechny buňky v tabulce, pokud se jedná o standardní buňku, tak vrátí $null$ a použije se předem definovaný typ sloupce.

Inicializace komponent uživatelského rozhraní provádějí třídy v $init$ balíčku na základě rolí a práv uživatele (obrázek č. 8). Toto je použito proto, aby logika inicializací na nebyla roztroušena po různých třídách uživatelského rozhraní, ale aby byla zapouzdřena v balíčku $init$.

Pokud uživatel může udělat nějakou akci, tak je pro ni vytvořen posluchač. Ten je vždy v samostatném balíčku zpravidla nazývaném „listener“.

Pro podporu přístupu více uživatelů najednou je použit návrhový vzor ThreadLocal. Ten umožňuje, aby každý připojený uživatel měl vlastní třídní proměnné v třídě $MyWiresApplication$ a jeho operace se prováděly ve vlastním vlákně.



Obrázek 8: Diagram vztahu inicializátorů a obrazovek systému.

Naše aplikace podporuje lokalizování, všechny texty, které zobrazuje GUI, nejsou přímo v kódu ale v samostatných souborech. Pro každý jazyk jeden soubor, který je umístěn v balíčku *resources*. Systém je lokalizován do českého a anglického jazyka. Podle nastavení lokalizace v prohlížeči se systém rozhodne jaký jazyk použít.

Rozhraní aplikace jsme se snažili udělat uživatelsky přívětivé a jednoduché. Vzhled komponent jsme upravovali kaskádovými styly (CSS). Máme vytvořené 3 různé barevně odlišná témata: modré, zelené a růžové. Implicitně je nastaveno modré ale uživatel v profilové obrazovce má možnost trvale si téma změnit.

Drželi jsme se hesla: „Pokud může uživatel něco udělat špatně, tak to dříve či později udělá“. Proto bychom uživateli neměli dovolit něco špatně udělat. Někdy pomohla i deaktivace některých tlačítek a podobně. Také všechny destruktivní změny v obrazovce MySocktes se musí potvrdit v dialogovém okně. Nebo při editování požadavků má uživatel možnost všechny změny zahodit a vrátit vše do původního stavu.

Pokrytí kódu testy je zde téměř nulové a pro budoucí vývoj tohoto systému by bylo dobré udělat testy i pro tento modul. Ovšem jednotkové testy se pro grafické rozhraní píšou obtížně.

6.4 Použití aplikačního rámce Spring

Aplikační rámec Spring je použit napříč všemi moduly, nejčastěji je využíváno jen jeho Core modul, který se stará o celý životní cyklus objektů (Bean). V modulech DAO-WS je využíván pro poskytování WS a definici koncových bodů (endpoint) a v Controlleru zase

pro WS klienta. V každém modulu jsou integrační testy kontrolující aplikační kontext Springu.

6.5 Práce v týmu

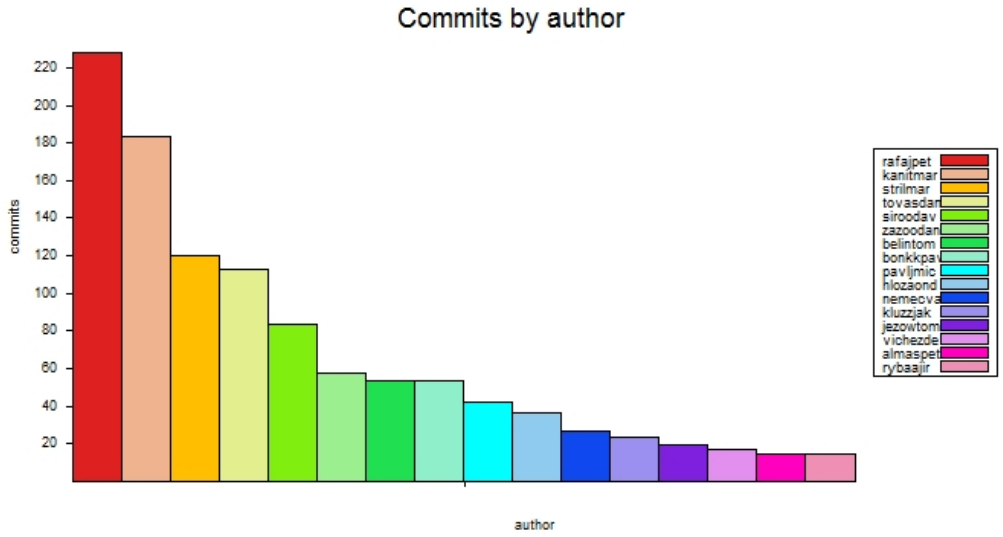
Pracovali jsme společně, pokud jsme měli nějaký úkol, rozdělili jsme ho na menší části a každý si vzal jednu. Někdo třeba dělal GUI někdo zase servisní metody. Nikdo v našem týmu se čistě nespecializoval na jednu věc třeba na databázi, ale každý jsme měli rozumět všemu. To odpovídá fungování týmu při agilních metodách vývoje, kdy všichni umí vše kvůli zastupitelnosti, ale každý je zároveň specialistou na jednu oblast a pokud se v ní objeví nějaký závažný problém tak jej řeší on. Mezi členy takového týmu patří jeden databázový specialista, analytik, architekt, konzultant se zákazníkem a tester. Pro takové projekty je ideální počet členů od pěti do patnácti. Pokud se používá nějaká technologie, tak alespoň jeden člověk má o ní hlubší znalosti a řeší s ní vzniklé problémy, které ostatní členové nebyli schopni zvládnout. Ale každý člen s ní umí pracovat. My jsme na vývoj byli tři vývojáři bez jakékoli předchozí zkušenosti s použitými technologiemi. Naštěstí první měsíc nám s projektem pomáhalo asi 10 zaměstnanců Tieta, kteří vytvořili základ architektury aplikace. Od prosince 2011 do února 2012 v našem týmu byli tři další vývojáři, kteří nás vedli při vývoji a pomáhali nám řešit vzniklé problémy. Bez nich bychom naši aplikaci nezvládli dokončit, za jejich pomoc jim děkuji.

6.6 Spoluautorství na projektu

Podílel jsem se na Use Case diagramech a doménovém modelu (obrázky č. 10, 11 a 1). Samostatně jsem vytvořil diagramy modulů, entit a incializátorů (obrázky č. 4, 2 a 8).

Spoustu času jsem věnoval implementaci diagramů Request Lifecycle (obrázky č. 6, 5) a chování obrazovek MySockets a Request. Nicméně vyjmenovat všechny části kódu, které jsem vytvořil, je zpětně velice složité. Proto přikládám alespoň obrázek č. 9 z SVN statistik, který ukazuje kolikrát jsem nahrával kód do repozitáře v porovnání s kolegy.

Celkový počet vytvořených tříd je 275, z toho bylo pro testy napsáno 59 tříd. Z XML definic je vygenerováno dalších 65 tříd pro webové služby.



Obrázek 9: Počet nahrání kódu do SVN („commit“). Můj sloupeček je třetí zleva.

7 Znalosti a dovednosti získané během studia a uplatněné v průběhu odborné praxe

V průběhu praxe mi přišla vhod celá řada znalostí zejména z předmětů Programovací jazyky I, kde jsem nabyl znalosti JavaSE. Dále pro specifikaci projektu jsme používali UML, se kterým jsem se seznámil v kurzu SWI. Nejcennější znalosti pro praxi jsem nabyl v předmětech Databázové a informační systémy a Vývoj informačních systémů, odkud jsem rozuměl problematice databází, objektově relačního mapování a návrhových vzorů pro tvorbu informačního systému. V oblasti databází jsem využil i znalosti z předmětu TZD. V rámci studia jsem získal zkušenosti s HTML a CSS, které jsem uplatnil při úpravách grafické podoby systému.

Důležitá byla i znalost angličtiny, jednak protože téměř všechny studijní materiály byly v angličtině a jednak také jednu dobu byl členem našeho týmu člověk, který nehovořil česky.

8 Znalosti a dovednosti scházející studentovi v průběhu odborné praxe

Chyběla mi celá řada znalostí týkajících se hlavně použitých technologií a také znalost JavaEE. Už na začátku odborné praxe se ukázalo, že nezapsat si předmět Java Technologie byla velká chyba. Nerozuměl jsem problematice vývoje informačních systémů v Javě. Také jsem se poprvé setkal s testováním softwaru pomocí Unit testů a frameworku Mockito. Musel jsem se naučit pracovat v týmu, což zahrnovalo například práci s verzovacím nástrojem SVN.

Nejobtížnější bylo zvládnutí aplikačních rámců Spring, Hibernate, Vaadin a Maven, se kterými jsem se setkal poprvé. Naštěstí díky výborné dokumentaci a také ochotě kolegů, jsem se s nimi naučil pracovat. Nicméně většina z aplikačních rámců a nástrojů má natolik specifické použití, že chápu, proč nejsou zahrnuty do bakalářského studijního programu.

9 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Jsem velice rád za to, že jsem měl možnost vykonat odbornou praxi jako ekvivalent bakalářské praxe. Odborná praxe mi přiblížila fungování firmy Tieto. Naučil jsem se používat výše zmíněné technologie alespoň na základní úrovni. Měl jsem možnost pracovat s řadou zajímavých lidí, od kterých jsem se toho spoustu naučil. Zjistil jsem, že při práci v týmu je důležité také umět dobře komunikovat. Zároveň je potřeba umět se rychle zorientovat v problému a přizpůsobit se.

Dosažené výsledky hodnotím jako vyhovující, podařilo se nám implementovat větší část systému. Projekt v této fázi je v omezené míře už použitelnou aplikací. Pro vývojáře, kteří budou pokračovat ve vývoji, je připravena dokumentace implementace, architektury a nasazení. Věřím, že naše aplikace bude pro chod firmy Tieto přínosná.

10 Reference

[1] Walls, Craig, Breidenbach, Ryan, *Spring in Action second Edition*, 730 stran, Greenwich: Manning Publications Co., 2007.

[2] Grönroos, Marko, *Book of Vaadin: 4th Edition*, 490 stran, 18. 4. 2012.

[3] *Maven Getting Started Guide*
URL: <http://maven.apache.org/guides/getting-started/index.html>
[cit. 2012-4-17].

[4] *Hibernate Documentation*
URL: <http://www.hibernate.org/docs>
[cit. 2012-4-21].

[5] Borovcová, Anna, *Testování webových aplikací Část II: Základy testování*, 42 stran.

[6] *Mockito*
URL: <http://code.google.com/p/mockito>
[cit. 2012-4-19].

[7] *SCRUM*
URL: http://www.zdrojak.cz/clanky/agilni-vyvoj-uvod/
[cit. 2012-4-18].

[8] Sutherland, Jeff, *Agile development: Lessons learned form the first SCRUM, 6 stran*, Říjen 2004

[9] *Agile Estimating Planning With Scrum*
URL: http://ahmedsamy.net/en/2011/06/16/agile-estimating-planning-with-scrum-presentation/
[cit. 2012-4-29].

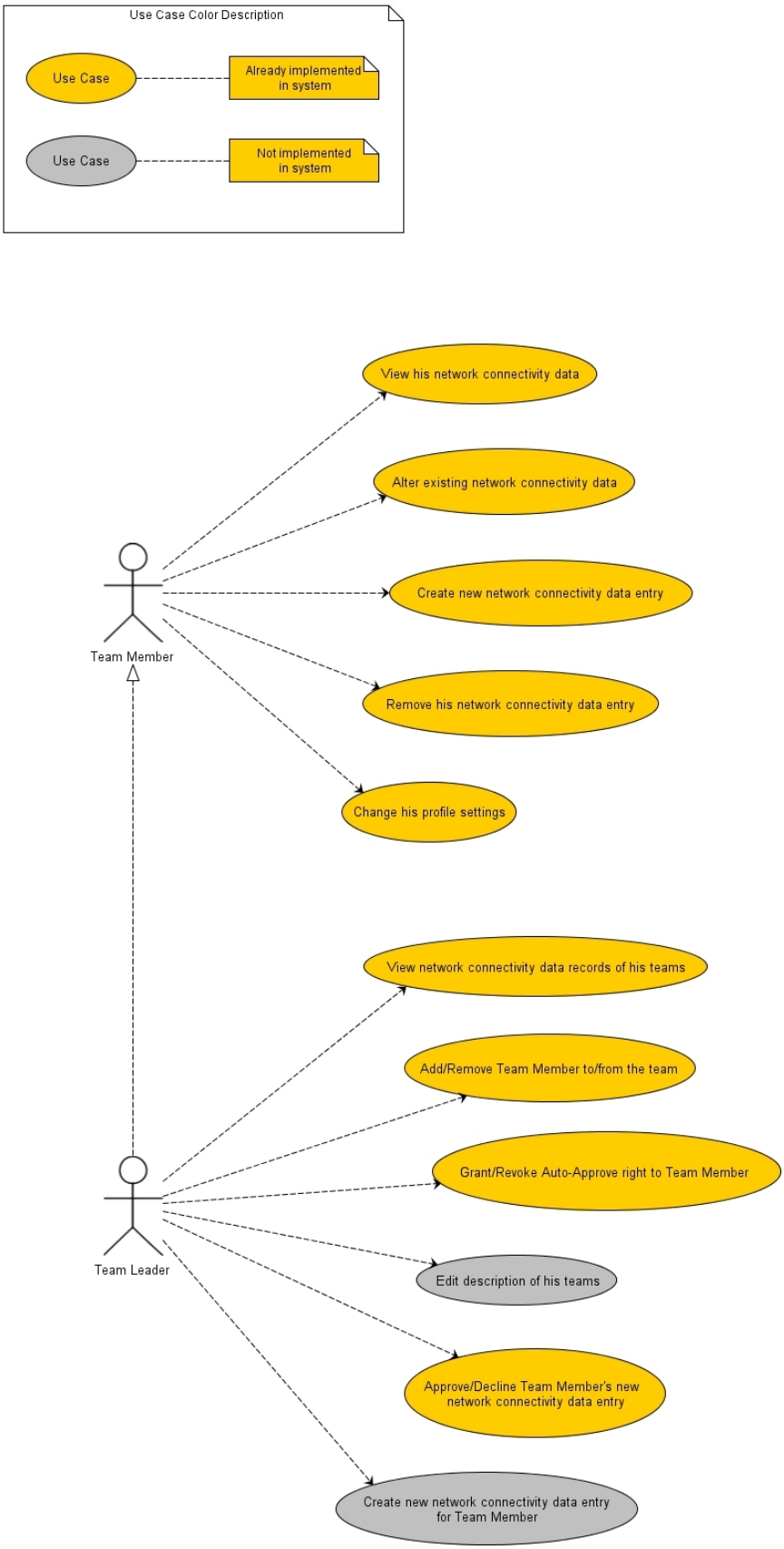
[10] *Web Services Glossary*
URL: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/
[cit. 2012-4-29].

[11] *Historie společnosti Tieto*
URL: http://www.tieto.cz/o-nas/historie/historie-spolecnosti-tieto
[cit. 2012-4-29].

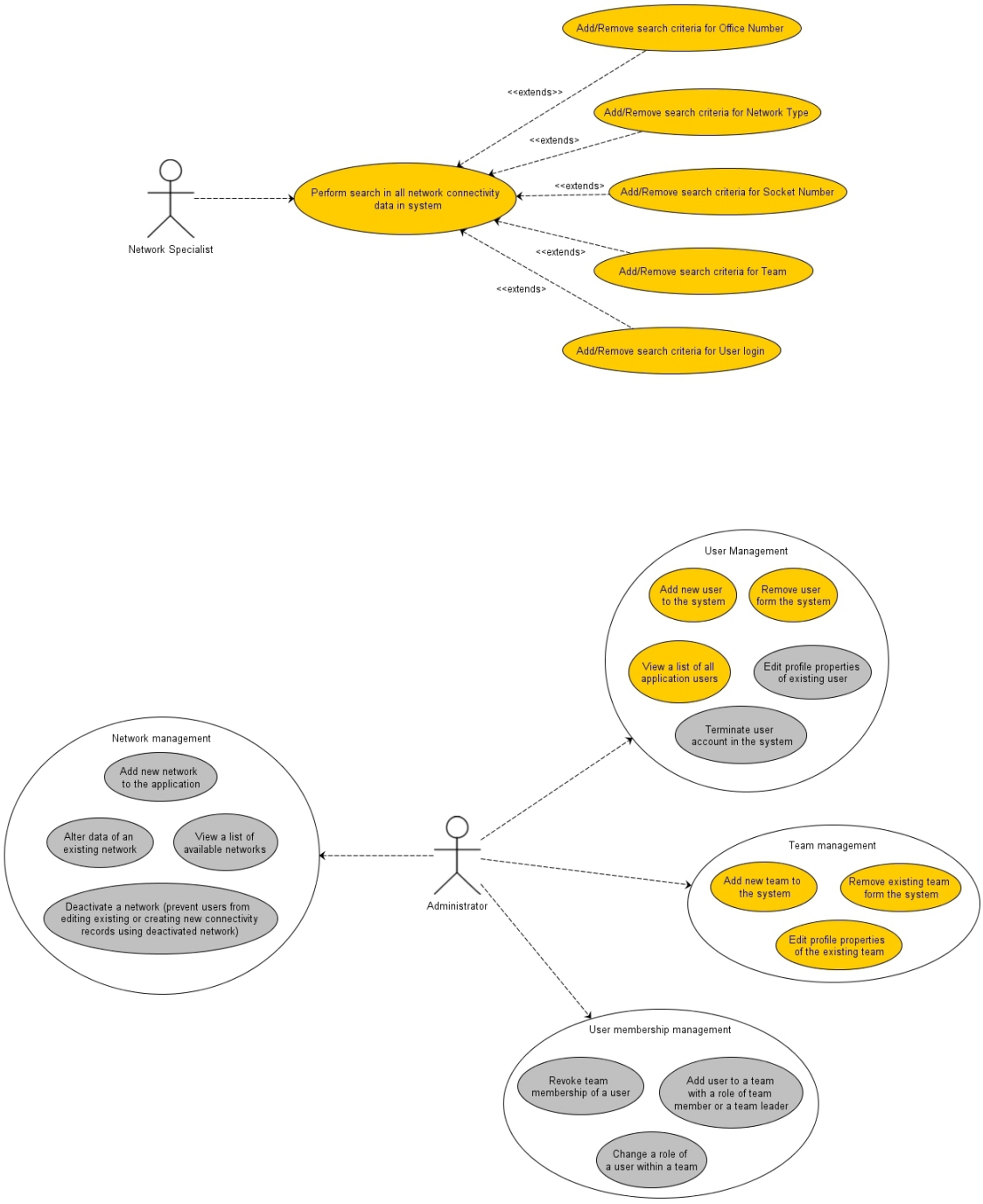
[12] *Data Access Object*
URL: http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html
[cit. 2011-11-05].

A Obrázky

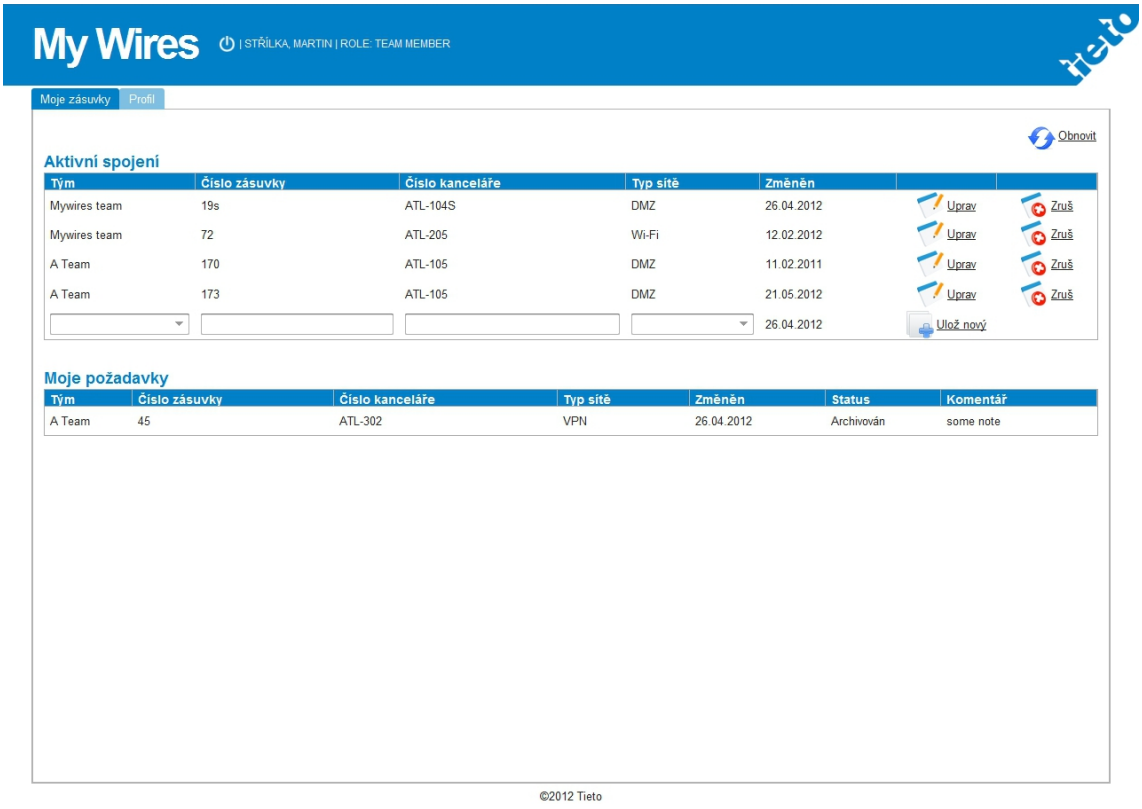
Zde jsou obrázky, které by svou velikostí rušily v textu bakalářské práce.



Obrázek 10: Use Case diagram pro role Team Member a Team Leader



Obrázek 11: Use Case diagram pro role Network Specialist a Administrátor



Obrázek 12: Ukázka GUI systému.